README

This folder contains a working, localized copy of Janusgraph that works without Elasticsearch mappings. This is the default version that has been downloaded from Janusgraph 0.6.3. This has minimal modification to run (that will be described below.

Modifications

bin/janusgraph-server.sh

Principally, the file bin/janusgraph-server.sh has been modified to not fork janusgraph process into the background. This is seen on line (modified lines in github) 211-214:

```
$JAVA -Dlog4j.configuration=$LOG4J_CONF $JAVA_OPTIONS -cp $CLASSPATH $JANUSGRAPH_SERVER_CMD "
PID=$!
disown $PID
echo $PID > "$PID_FILE"
```

which has been modified to:

```
$JAVA -Dlog4j.configuration=$LOG4J_CONF $JAVA_OPTIONS -cp $CLASSPATH $JANUSGRAPH_SERVER_CMD "
PID=$!
#disown $PID
echo $PID > "$PID_FILE"
```

this is not a functional change, this simply prevents the server from being forked into the background. This is good for monitoring in docker, but is not strictly required if not deploying within dockerized process.

gremlin-server.yml

The following file has been created which is the default configuration for the janusgraph server (key values shown here, see conf/gremlin-server/gremlin-server.yml for more detail):

```
host: 0.0.0.0
port: 8182
evaluationTimeout: 30000
channelizer: org.apache.tinkerpop.gremlin.server.channel.WebSocketChannelizer
graphManager: org.janusgraph.graphdb.management.JanusGraphManager
graphs: {
   ConfigurationManagementGraph: conf/janusgraph-cql-configurationgraph.properties
}
```

This configuration configures janusgraph to run on the local host (0.0.0.0) at port 8182. This is configured as a configurationgraph which allows multiple graphs to be stored in one janusgraph instance (the recommended way to run janusgraph, see Configured Graph Factory The actual connection information to CQL is stored in `conf/janusgraph-cql-configurationgraph.properties`.

conf/janusgraph-cql-configurationgraph.properties

(see `conf/janusgraph-cql-configurationgraph.properties` for more detail) This config file dictates the specific connection information to cassandra for the ConfigurationManagementGraph. This does not setup a graph, but rather sets up a keyspace for management of multiple graphs (stored in other keyspaces). This allows the flexibility to manage multiple graphs at once stored in different configuration, which is very helpful for testing different storage methodologies. The following are key configurations used in this file that were modified:

```
storage.backend=cql
storage.username=cassandra
storage.password=cassandra
storage.hostname=janusgraph_cassandra
storage.cql.keyspace=janusgraph
```

for testing, username, password, and hostname will dictate connection to cql.

Running

Outside of docker

If running outside of docker, this assumes that a cassandra server is running at a path accessible from the local host. If using a different username/password/hostname than those specified above, update `conf/janusgraph-cql-configurationgraph.properties` with your respective connection properties

Inside of docker

If running within a docker instance, a docker-compose has been provided that will work out of the box with these configuration files. Simply run: `docker-compose up -d` which will build janusgraph and run dockerized cassandra.

Graph Setup

This section will describe how to setup the ConfiguredGraphFactory, create a graph, and load the GraphOfTheGods (Toy Example Graph) for testing. This will assume that you are using the dockerized version. If running outside of docker, it is assumed that this command is currently running: `./bin/janusgraph-server.sh start conf/gremlin-server/gremlin-server.yml` The next three sections are contained in `GraphSetup.groovy` and can be run sequentially within the gremlin console. A clipping of the console is provided in `CommandExample.txt` so that a developer can quickly step through the code if desired.

ConfiguredGraphFactory Template Setup

This is optional, but recommended. The template configuration holds basic information about your graphs that can be re-used when setting up new graphs. `docker exec -it janusgraphbuild /app/bin/gremlin.sh` then run the following in the gremlin-console:

```
:remote connect tinkerpop.server conf/remote.yaml session
:remote console
map = new HashMap<>()
map.put('storage.backend', 'cql')
map.put('storage.hostname', 'cassandra')
map.put('storage.username', 'cassandra')
map.put('storage.password', 'cassandra')
map.put('storage.cql.keyspace', 'janusgraph-configuredgraphmanangement')
map.put('schema.default', 'none')
map.put('storage.batch-loading',true)
map.put('storage.buffer-size',10000)
conf = new MapConfiguration(map)
ConfiguredGraphFactory.createTemplateConfiguration(conf)
```

Create ToyGraph

This is required, and will need to be modified if a configured graph factory is not used.

```
map = ConfiguredGraphFactory.getConfiguration()
map.put('storage.cql.keyspace', 'graphs_janusgraph_toygraph')
map.put('graph.graphname', 'toygraph')
conf = new MapConfiguration(map)
ConfiguredGraphFactory.createConfiguration(conf)
```

Load graph of the gods

Graph of the gods is a very simple graph provided by janusgraph to let users test the functionality of the code. It is very small, but useful for practicing gremlin. Note, this does not use elasticsearch, so these queries (such as `.has`) only functionally work because of how small it is.

```
mgmt = ConfiguredGraphFactory.open('toygraph').openManagement()
graph = ConfiguredGraphFactory.open('toygraph')
GraphOfTheGodsFactory.loadWithoutMixedIndex(graph,true)
```

Gremlin Example

Now that we have the GraphOfTheGods loaded, we can test some gremlin queries such as:

```
graph = ConfiguredGraphFactory.open('toygraph')
gremlin> g = graph.traversal()
==>graphtraversalsource[standardjanusgraph[cql:[cassandra]], standard]
gremlin> g.V().has('name', 'saturn').valueMap()
==>{name=[saturn], age=[10000]}
gremlin> g.V().bothE().limit(10).valueMap()
==>{reason=loves waves}
==>{}
==>{}
==>{}
==>{}
==>{}
==>{}
==>{place=POINT (23.9 37.700001), time=2}
==>{time=12, place=POINT (22 39)}
==>{place=POINT (23.700001 38.099998), time=1}
gremlin> neptune_id = g.V().has('name', 'neptune').id().next()
==>12304
gremlin> g.V(neptune_id).outE('brother').inV().inE('father').outV().out().has('name', 'nemean
==>path[v[12304], e[b2a-9hs-b2t-6c0][12304-brother->8208], v[8208], e[363-388-6c5-6c0][4184-f
```